ATTORNEY DOCKET NO: 06502.0115-00000

United States Patent Application of

KEN ARNOLD, JIM WALDO, ANN WOLLRATH, AND PETER JONES

FOR

METHOD AND APPARATUS FOR DYNAMIC DISTRIBUTED COMPUTING OVER A NETWORK

BACKGROUND OF THE INVENTION

INS A1

Field of the Invention

This invention generally relates to distributed computing systems and more particularly, to a method and apparatus for performing dynamic distributed computing over a network.

Description of the Related Art

In a distributed computing network, users can harness the processing capabilities of numerous computers coupled to the network. Tasks with many different independent calculations can be quickly processed in parallel by dividing the processing among different computers on the network. Further, specialized tasks can be computed more quickly by locating a computer on the network most suitable for processing the data. For example, a task executing on a client system which performs an intense floating point calculation may execute faster on a server system coupled to the network which has specialized floating point hardware suitable for the particular calculations.

Unfortunately, conventional techniques for distributed computing are not easily implemented in the typical heterogenous computing environments. Each computer on the network is typically heterogeneous containing different processor and operating system combinations, and require different object modules for execution. On the client side, different object modules requires that the user compiles different versions of the task for each different platform and loads the module onto the corresponding platform adding storage requirements to each client and also

requiring porting and compiling the same tasks multiple times. Further, conventional techniques require that the code be distributed over the computers well before the code is executed. In the conventional systems, the extensive preparation required for performing distributed computing deterred many from exploiting this technology.

Distributed computing systems based on scripting languages are an improvement over some conventional distributed computing systems. Unfortunately, scripting based systems eliminate the need to recompile code, but are still very inefficient. A scripting based distributed system can execute the same instructions on multiple platforms because the language is interpreted by an interpreter located on each system. Consequently, most scripting languages are slow since they must translate high level scripting instructions into low level native instructions in real time. Moreover, scripting languages are hard to optimize and can waste storage space since they are not generally compressed.

Based on the above limitations found in conventional systems, it is desirable to improve distributed computing systems.

SUMMARY OF THE INVENTION

In one aspect of the present invention associated with a client computer, a method and apparatus for dynamic distributed computing is provided. Initially, the client selects a server from the network to process the task. This selection can be based on the availability of the server or the specialized processing capabilities of the server. Next, a client stub marshals the parameters and data into a task request. The client sends the task request to the server which invokes a generic compute method. The server automatically determines if the types associated with the task are available on the server and downloads the task types from the network as necessary. Information in the task types are used to extract parameters and data stored in the particular task request. The generic compute method is used to execute the task request on the selected server. After the server processes the task request, the client receives the results, or the computed task, back from the selected server.

In another aspect of the present invention associated with a server computer, a method and apparatus for dynamic distributed computing is provided. Initially, the server will automatically determine which task types are available on the server and will download task types from the network as necessary. These task types help the server unmarshal parameters and data from a task request and generate a local task. Next, the server invokes a generic compute method capable of processing all types of compute tasks or subtypes of a compute task. The generic compute method is used to execute the task request on the selected server. If a subsequent task will use the

results, the server stores the results from the computed tasks in a local cache. Once the task has completed, the server returns the results, or the computed task, to the client.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an embodiment of the invention and, together with the description, serve to explain the advantages, and principles of the invention.

In the drawings:

FIG. 1. illustrates a network suitable for use with methods and systems consistent with the present invention;

FIG. 2 is block diagram of a computer system suitable for use with methods and systems consistent with the present invention;

FIG. 3 is a block diagram representation of a client-server networking environment suitable for use with methods and systems consistent with the present invention;

FIG. 4 is a flow chart of the steps a client performs in accordance with methods and systems consistent with the present invention; and

FIG. 5 is a flow chart the steps performed by a server in accordance with methods and systems consistent with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

INTRODUCTION

Reference will now be made in detail to an implementation of the present invention as illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings and the following description to refer to the same or like parts.

Systems consistent with the present invention address shortcomings of the prior art and provide a dynamic distributed computing system used over a network of server computers. This dynamic distributed computing system is particularly useful in heterogenous computer networks having computers with different processors, different operating systems, and combinations thereof. Such a system allows a client application to select a server computer at runtime to execute a particular task. In method and systems consistent with the present invention, the task is an object having a particular type or class definition. The server can generally defer knowing the actual class definition until the parameters and data associated with the object task are received on the server. Consequently, the particular type is downloaded by the server if it is not available on the server. For example, if an object instance of an unknown class is transmitted to the server, the server downloads the unknown class. The server then uses this class to process the object. This late association of a class definition to an object increases the flexibility in processing complex tasks over a network of server computers. Further, the present design facilitates this flexibility with minimal additional overhead by utilizing features in existing remote procedure call subsystems

such as the Remote Method Invocation (RMI) subsystem developed by Sun Microsystems, Inc. of Mountain View, California. For more information on Remote Method Invocation (RMI) see co-pending U.S. Patent Application, "System and Method For Facilitating Loading of "Stub" Information to Enable a Program Operating in One Address Space to Invoke Processing of a Remote Method or Procedure in Another Address Space" having serial number 08/636,706, filed April 23, 1996 by Ann M. Wollrath, James Waldo, and Roger Riggs, assigned to a common assignee and hereby incorporated by reference. Also, RMI is also described in further detail in the RMI specification on the JavaSoft WebPage at FTP://ftp.javasoft.com/docs/jdk1.2/rmi-spec-jdk1.2.ps, which is also hereby incorporated by reference.

Unlike conventional systems, a task in the dynamic distributed system consistent with the present invention can be written once and executed on any server computer in a network. This capability is particularly advantageous in a heterogeneous network because the task does not have to be ported to every platform before it is executed. Instead, a generic compute task designed in accordance with the present invention is loaded on each system. This generic compute task is capable of executing a wide variety of tasks specified by the client at runtime. For example, one can develop a type called "Compute" and a generic compute task which accepts the "Compute" type in an object-oriented language, such as Java. Java is described in many texts, including one that is entitled "The Java Language Specification" by James Gosling, Bill Joy, and Guy Steele, Addison-Wesley (1996), which is hereby

incorporated by reference. The client creates a task having a subtype of the type "Compute" and passes an object corresponding to task to the generic compute task on the server. A remote procedure call mechanism downloads the object to the server and the generic compute task which executes the task.

In Java, the task transmitted by the client is actually an object including a series of bytecodes. These bytes codes can be executed immediately as long as the server implements a Java Virtual Machine (JVM). The JVM can be implemented directly in hardware or efficiently simulated in a software layer running on top of the native operating system. The Java language was designed to run on computing systems with characteristics that are specified by the Java Virtual Machine (JVM) Specification. The JVM specification is described in greater detail in Lindholm and Yellin, The Java Virtual Machine Specification, Addison-Wesley (1997), which is hereby incorporated by reference. This uniform JVM environment allows homogeneous execution of tasks even though the computer systems are heterogenous and have different processors, different operating systems, and combinations thereof. Combining a powerful remote procedure call subsystem with a generic compute task on the server, designed in accordance with the present invention, results in a powerful dynamic distributed computing environment.

A compute server using bytecodes can process a task much faster than systems using conventional text based scripting languages or other character based languages. Each bytecode is compact (8 bits) and is in a numeric format. Consequently, the server computer does not spend compute cycles parsing the characters and arguments

at run time. Also, the bytecodes can be optimized on the client before transporting them to the server. The server optionally can convert the bytecodes to native instructions for execution directly on the hardware at run time using a processing mechanism such as a Just-in-Time (JIT) compiler. For more information on JIT compilers see The Java Virtual Machine Specification.

A system designed in accordance with the present invention assumes that each client is capable of communicating to each server over a common networking protocol such as TCP/IP. Also, it is assumed that there is a remote procedure call (RPC) subsystem on the client and server which is capable of receiving remote requests from a client and executing them on the server. This RPC system also automatically downloads code and related information needed for performing the task at run time. RMI developed by Sun Microsystems, Inc. is a suitable RPC subsystem providing these features. One skilled in the art, however, will appreciate that other RPC subsystems, such as DCOM/COM from Microsoft, Inc., may be used in lieu of RMI.

COMPUTER NETWORK

Fig. 1 illustrates a network 100 in which one embodiment of the present invention can be implemented. Network 100 includes Local Area Network (LAN) 101, backbone or Wide Area Network (WAN) 112, and Local Area Network (LAN) 116 in its essential configuration. LAN 101 includes a series of work stations and server computers 102, 104, 106, and 108. LAN 116 includes a series of work

stations and server computers 118, 120, 122, and 124. These computer systems 102-108 and 118-124 are coupled together to share information, transmit data, and also share computational capabilities. LAN 101 is coupled to the larger overall network using a network interconnect device 110. The specific type of network interconnect device can be a router, a switch, or a hub depending on the particular network configuration. In general, network interconnect device 110 includes routers, switches, hubs or any other network interconnect device capable of coupling together a LAN 101, a WAN 112, and LAN 116 with user terminals into an integrated network. Network interconnect device 114 can also include routers, switches, hubs, or any other network interconnect device capable of coupling the computers on LAN 116 with user terminals into an integrated network. In general, a dynamic distributed computing system designed in accordance with the present invention is typically located on each computer system coupled to network 100 . Accordingly, each computer may operate as either a client or a server depending on the particular request being made and the services being provided. Typically, the client requests that a task is computed on a server computer and the server computer will process the task.

COMPUTER SYSTEM

Referring now to Fig. 2, the system architecture for a computer system 200 suitable for practicing methods and systems consistent with the present invention is illustrated. The exemplary computer system 200 is for descriptive purposes only. Although the description may refer to terms commonly used in describing particular

computer systems, such as in IBM PS/2 personal computer, the description and concepts equally apply to other computer systems, such as network computers, workstation, and even mainframe computers having architectures dissimilar to Fig. 1.

Furthermore, the implementation is described with reference to a computer system implementing the Java programming language and Java Virtual Machine specifications, although the invention is equally applicable to other computer systems having similar requirements. Specifically, the present invention may be implemented with both object-oriented and nonobject-oriented programming systems.

Computer system 200 includes a central processing unit (CPU) 105, which may be implemented with a conventional microprocessor, a random access memory (RAM) 210 for temporary storage of information, and a read only memory (ROM) 215 for permanent storage of information. A memory controller 220 is provided for controlling RAM 210.

A bus 230 interconnects the components of computer system 200. A bus controller 225 is provided for controlling bus 230. An interrupt controller 235 is used for receiving and processing various interrupt signals from the system components.

Mass storage may be provided by diskette 242, CD ROM 247, or hard drive 252. Data and software may be exchanged with computer system 200 via removable media such as diskette 242 and CD ROM 247. Diskette 242 is insertable into diskette drive 241 which is, in turn, connected to bus 230 by a controller 240. Similarly, CD ROM 247 is insertable into CD ROM drive 246 which is, in turn, connected to bus

230 by controller 245. Hard disk 252 is part of a fixed disk drive 251 which is connected to bus 230 by controller 250.

User input to computer system 200 may be provided by a number of devices. For example, a keyboard 256 and mouse 257 are connected to bus 230 by controller 255. It will be obvious to those reasonably skilled in the art that other input devices, such as a pen and/or tablet may be connected to bus 230 and an appropriate controller and software, as required. DMA controller 260 is provided for performing direct memory access to RAM 210 A visual display is generated by video controller 265 which controls video display 270.

Computer system 200 also includes a communications adaptor 290 which allows the system to be interconnected to a local area network (LAN) or a wide area network (WAN), schematically illustrated by bus 291 and network 295.

Operation of computer system 200 is generally controlled and coordinated by operating system software. The operating system controls allocation of system resources and performs tasks such as processing scheduling, memory management, networking, and services, among things.

DYNAMIC DISTRIBUTED COMPUTING

Dynamic distributed computing is generally a client server process. The client-server relationship is established for each call being made and generally the roles can change. Typically, the client is defined as the process making a call to request resources located or controlled by the server. In this context, the computer or

processor executing the requesting process may also be referred to as a client. However, these roles may change depending on the context of information and particular processing which is taking place.

Fig. 3 is a block diagram representation of a client-server networking environment used to implement one embodiment of the present invention. This diagram includes those subsystems closely related to the present invention to emphasize one embodiment of the present invention. Additional subsystems, excluded in Fig. 3, may be necessary depending on the actual implementation.

Accordingly, Fig. 3 includes a client 302, a server 316, and an object/method repository 314 which are all operatively coupled to a network 312. Client 302 includes an application 304 which makes a remote compute call 306 to process a task on a remote server computer. A remote stub 310, typically generated using a remote procedure call subsystem, as described in the RMI specification, is used to package parameters and data associated with the specific remote compute call 306. The typical client can also includes a collection of local objects/methods 308 which may contain the type of task client 302 calls remote compute call 306 to execute. Alternatively, the tasks can be located in object method repository 314 and are accessed by compute method 320 as needed. Server 316 includes a remote skeleton 322 to unmarshal the parameters and data transmitted from the client. Remote skeleton 322 prepares information for use by compute method 320. A local objects/methods 324 also includes tasks client 302 can ask the server 316 to process.

In operation, remote compute call 306 makes a call to a compute method 320 to process a particular task. A remote stub 310 marshals information on the calling method so that a compute method 320 on server 316 can execute the task. Remote stub 310 may also marshal basic parameters used as arguments by compute method 320 on server 302. Remote skeleton 322 receives the task and unmarshals data and parameters received over the network and provides them to compute method 320. If the task and related types are not available on server 316, the skeleton downloads the types from client 302, object/method repository 314, or some other safe and reliable source of the missing types. The type information maps the location of data in the object and allows the remote skeleton to complete processing the object. RMI (not shown) is one remote procedure call (RPC) system capable of providing remote stub 310 and remote skeleton 322. Once the object is processed by the skeleton, compute method 320 executes the task and returns the computed task or computed task results to client 302.

Fig. 4 is a flow chart of the steps performed by a client when utilizing the dynamic distributed computing system and method consistent with the present invention. Initially, the client selects a suitable server from the network to process the task (step 402). The selection criteria can be based upon the overall processing load distribution among the collection of server computers or the specialized computing capabilities of each server computer. For example, load balancing techniques may be used to automatically determine which computer has the least load at a given moment. Further, some computers having specialized hardware, such as graphic

accelerators or math co-processors, may be selected by the client because the task has intense graphic calculations, such as rendering three dimensional wireframes, or must perform many floating point calculations.

Once the server is selected, the client invokes a remote compute method on the selected server (step 404). An RPC system, such as RMI, facilitates invoking the remote compute method on a server computer. Typically, the client need only know that the remote compute method can be used as a conduit to process a particular task on a remote computer. For example, in Java the remote instruction "Server.runTask(new PI(1000))" executed on a client causes a remote method "runTask" to be invoked on a remote server "Server" of type "ComputeServer". This step provides the task (in this case the task is a type task object instantiated by the "new PI(1000)) as a parameter to the generic compute method through the remote method "runTask". The "runTask" method on the server implements a Compute remote interface. Optionally, this instruction can indicate to the server that results from the computed task should be stored in a result cache on the selected server. This enables subsequent tasks to share the results between iterations. For example, the results from calculating "PI" may be used later by another remote method to compute the volume of a sphere or perform another precise calculation using the value of "PI".

A stub is used to marshal parameters and data into a task request. The task request is then provided to the selected server. Typically, the task request includes data and parameters for the task as well as a network location for the type or class if it is not present on the server. A skeleton on the server uses the type or class

information to process the object and unmarshall data and parameters. In a system using Java and RMI, the task request is an object and the class location information is contained in a codebase URL (universal record locator) parameter. Further details on this are contained in the RMI Specification. The server can schedule the task for execution immediately or whenever the server finds a suitable time for executing the task. After the server performs the computation, the client receives the results from the computed task (step 408).

Fig. 5 is a flow chart of the steps performed by the dynamic distributed computing system and methods consistent with the present invention. Initially, a skeleton on the server unmarshalls parameters and data from a task request and recreates the original task as transmitted (step 504). Unmarshalling these parameters may include downloading several additional types. The skeleton determines if the types related to the task request are available on the server (step 506). If the types associated with the task request are not available, the skeleton must download the tasks from one of the areas on the network (step 509). For example, if a "PI()" class is not on the server, the skeleton server will down load this type from the client. The type or class is used by the skeleton to map data in the object and marshall parameters and data.

Typically, the client will indicate in the request package where the particular type is located. The skeleton can download the requested type from a object/method repository and can cache the type for future server requests. Also, the requested type could also be located on the client. For example, in Java and RMI the class

containing the particular type is located in the given codebase URL (universal record locator) transmitted by the client. Dynamic class loading features in RMI facilitate the automatic downloading of the class using the codebase. These types enable the skeleton to parse the task request and extract the appropriate data and parameters. The steps outlined above make the parameters and data readily available for further processing.

Once the appropriate types are available, the skeleton invokes the generic compute method (step 508). The generic compute method on the server then executes the specific task requested by the client (step 510). For example, assume the client calls “ComputeServer.runTask(new PI(1000))”. The skeleton will invoke the generic compute method “runTask” on the server. The "runTask" method calls the "run()" method embedded in the task called by the client. Further, the "runTask' method implements the remote interface "Compute" which maintains the remote connection with the client. At the option of the client or a predetermined setting on the server, the skeleton stores results from the computed tasks in a cache if a subsequent task will use the results. As a final step on the server, the computed task or results are returned to the client by executing "return t.run()" on the server (step 512).

EXEMPLARY IMPLEMENTATION

Consistent with the present invention, the following code sample is provided as one implementation. Although this example is provided in the object-oriented Java

programming language other programming languages could also be used. For example, the server can include the following Java code:

```
THE TASK
public interface Task extends Serializable {
                                        //This interface allows a class (the "PI"
                                        // class ) to implement the abstract
                                        // run() class
{
        Public Object run();
}


THE REMOTE INTERFACE:
import java.rmi.*;
public interface Compute extends Remote {
                                                  // The RMI/RPC Interface
public Object runTask(Task t) throws RemoteException;
                                                  //The abstract runIt method
}
```

THE COMPUTE SERVER IMPLEMENTATION

```
import java.rmi.*;
import java.rmi.server.*;
public class ComputeServer extends UnicastRemoteObject
implements Compute{
        public ComputeServer () throws RemoteExecption{}
                                        //Implements the Compute interface
                                        //abstract "runTask" method
// ... Code in this area is used for initializing the routine with RPC system
        public Object runTask (Task t) throws RemoteException
                                        // runTask implements the abstract
method
                                        // defined in ComputerServer interface
return t.run();         //
}
```

The following exemplary Java code can be used on a client performing dynamic distributed computing consistent with the present invention.

```
class PI {
private int precision;
```

```
PI (int howManyPlaces) { // sets precision of PI value to be calculated later
        precision = howManyPlaces;
}
public Object run() {  // implement the abstract run method in the
                       // compute interface
double pi = computePIsomehow(precision); // calcualate pi
return new Double(pi);
}

public static void main (String[] args) {
ComputerServer server = getAComputerServer();    // Select a server from
                                                 // the network
                                                 // and store in remote
                                                 // compute call to RMI
                                                 // RPC abstract remote
                                                 // interface
Double pi = server.runTask(new PI(1000)); // implement abstract remote
                                          // to execute a "pi"computation
                                          // defined in "PI"class.
System.out.println("PI seems to be "+pi);  // return results in "pi" variable
                                           // and print to standard out
```

While specific embodiments have been described herein for purposes of illustration, various modifications may be made without departing from the spirit and scope of the invention. Those skilled in the art understand that the present invention can be implemented in a wide variety of hardware and software platforms and is not limited to the traditional routers, switches, and intelligent hub devices discussed above. Accordingly, the invention is not limited to the above described embodiments, but instead is defined by the appended claims in light of their full scope of equivalents.